09728693.120100

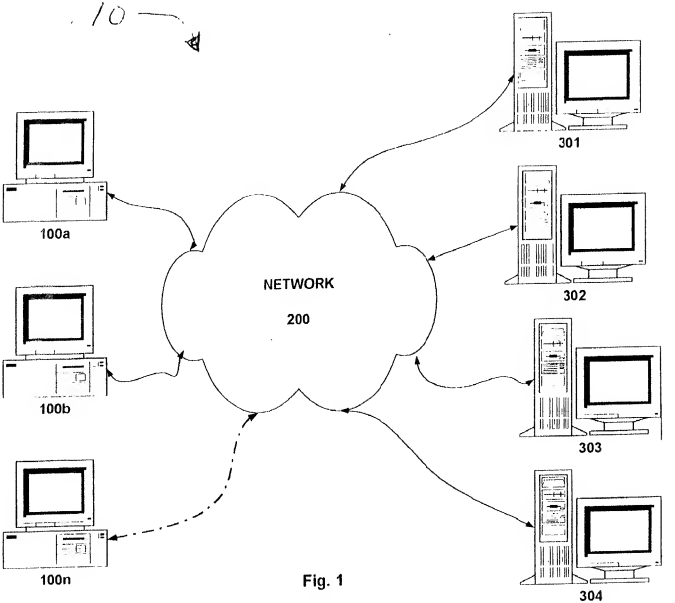

Fig. 1

09728693.120102

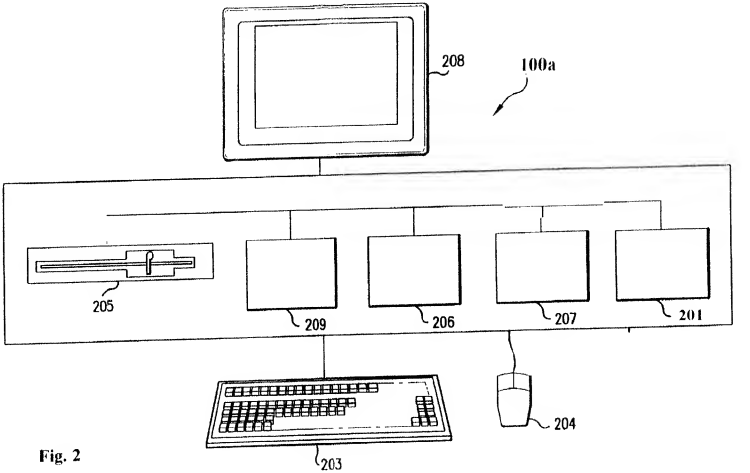

Fig. 2

09728693.120100

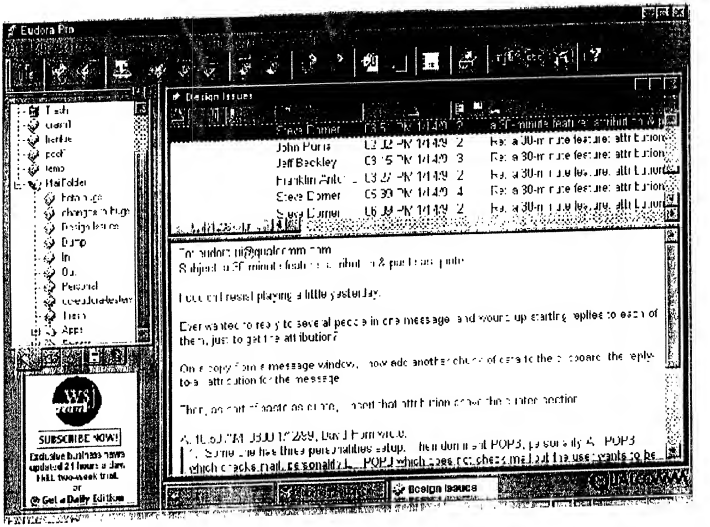

Fig. 3A

00702E 69982760

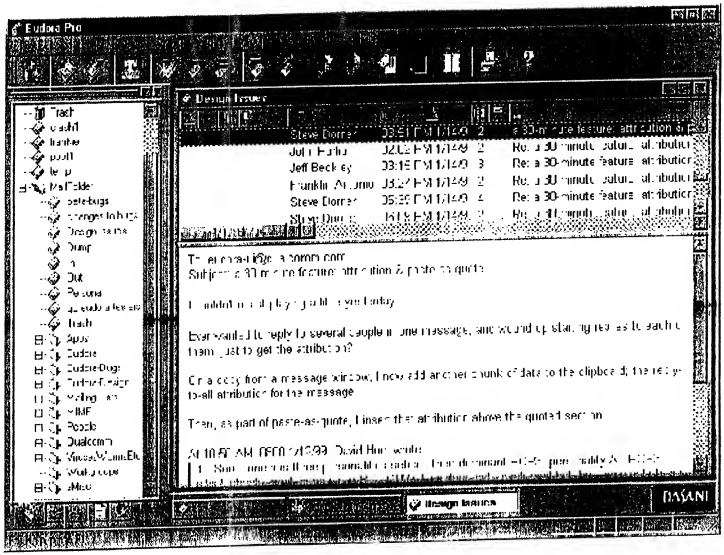

Fig. 3B

09/28/2001 15:02:26

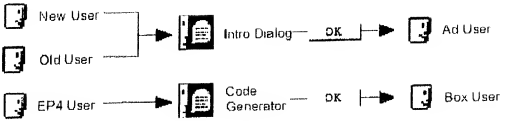

Fig. 4A

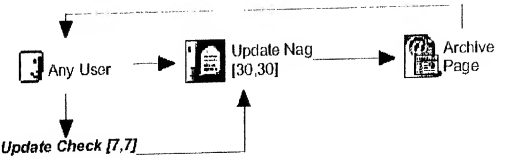

Fig. 7A

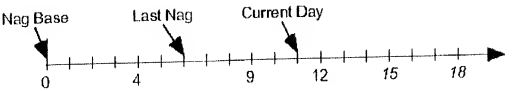

Fig. 11

09282693.120100

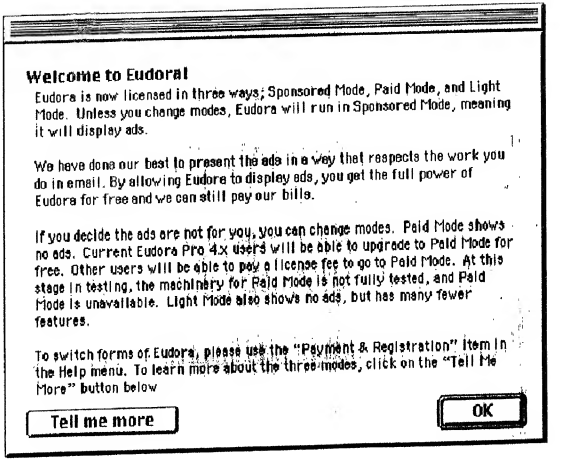

Fig. 4B

00102T*E6983260

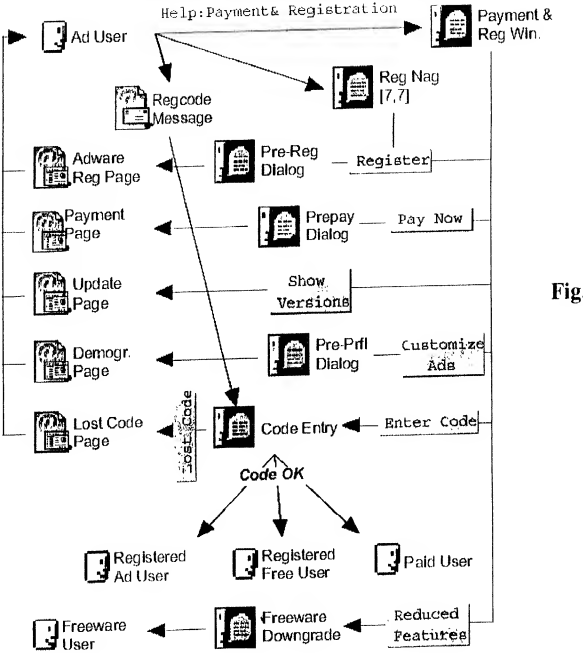

Fig. 5A

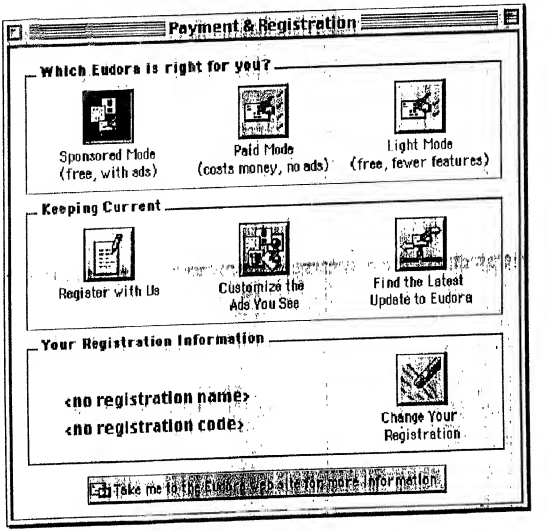
Payment & Registration
Which Eudora is right for you?
Sponsored Mode
(free, with ads)
Paid Mode
(costs money, no ads)
Light Mode
(free, fewer features)
Keeping Current
Register with Us
Customize the
Ads You See
Find the Latest
Update to Eudora
Your Registration Information
<no registration name>
<no registration code>
Change Your
Registration
Take me to the Eudora Web site for more information

Fig. 5B

09728693.120100

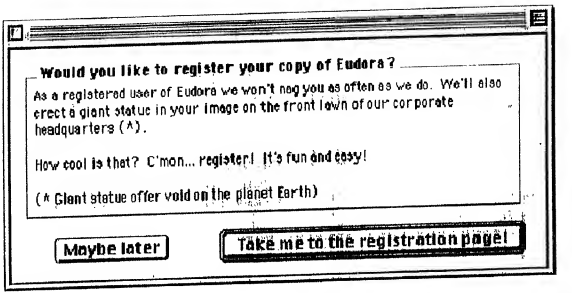

Fig. 5C

09/28/93 - 130100

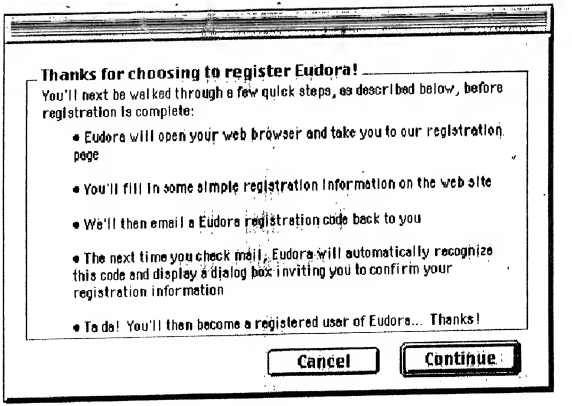

Fig. 5D

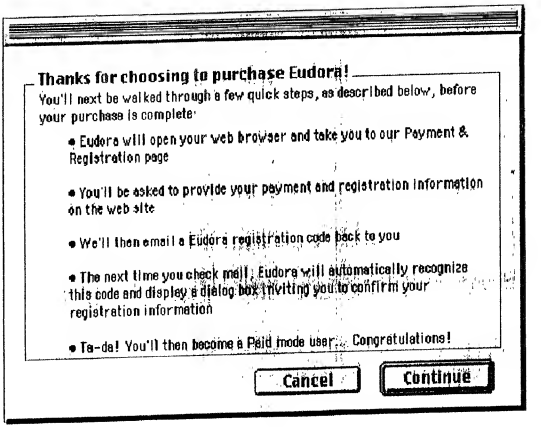
Thanks for choosing to purchase Eudora!
You'll next be walked through a few quick steps, as described below, before your purchase is complete:
• Eudora will open your web browser and take you to our Payment & Registration page
• You'll be asked to provide your payment and registration information on the web site
• We'll then email a Eudora registration code back to you
• The next time you check mail, Eudora will automatically recognize this code and display a dialog box inviting you to confirm your registration information
• Ta-da! You'll then become a Paid mode user... Congratulations!
Cancel
Continue

Fig. 5E

09728653 .120102

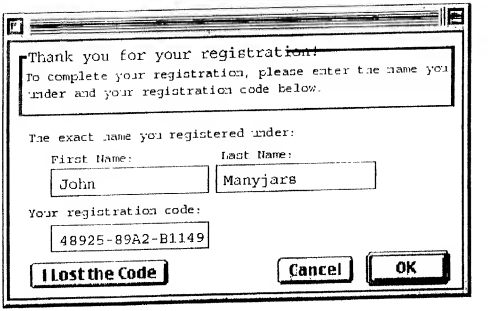

Fig. 5F

09728693.120100

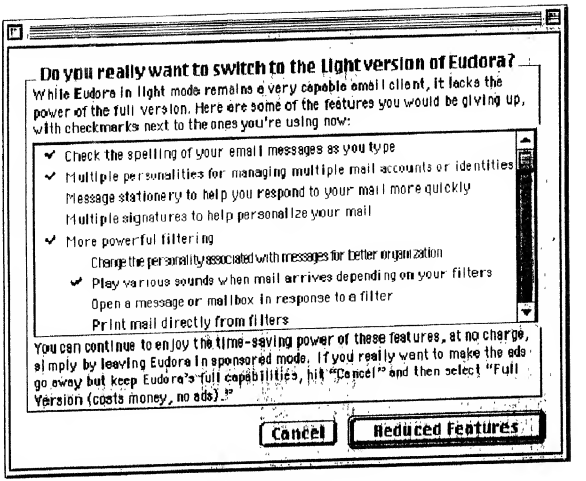

Fig. 5G

00102T'E6982466

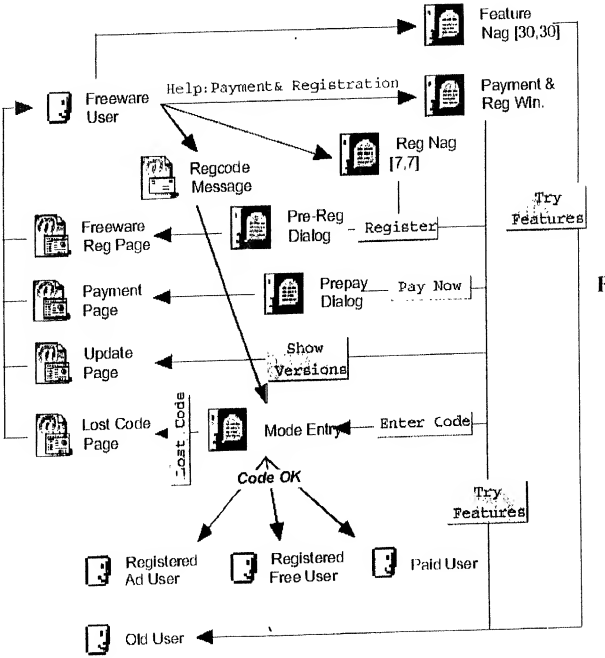

Fig. 6A

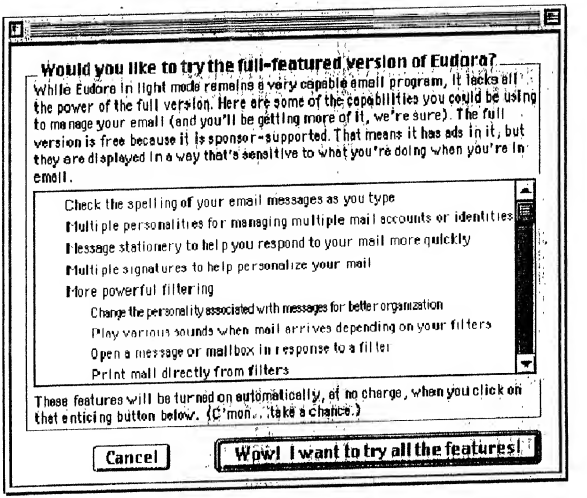
Would you like to try the full-featured version of Eudora?
While Eudora in light mode remains a very capable email program, it lacks all the power of the full version. Here are some of the capabilities you could be using to manage your email (and you'll be getting more of it, we're sure). The full version is free because it is sponsor-supported. That means it has ads in it, but they are displayed in a way that's sensitive to what you're doing when you're in email.
Check the spelling of your email messages as you type
Multiple personalities for managing multiple mail accounts or identities
Message stationery to help you respond to your mail more quickly
Multiple signatures to help personalize your mail
More powerful filtering
Change the personality associated with messages for better organization
Play various sounds when mail arrives depending on your filters
Open a message or mailbox in response to a filter
Print mail directly from filters
These features will be turned on automatically, at no charge, when you click on that enticing button below. (C'mon... take a chance.)
Cancel
Wow! I want to try all the features!

Fig. 6B

DOTO2F"E0982760

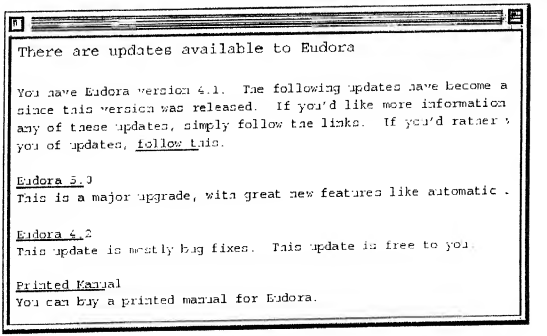

Fig. 7B

09282693.1201.00

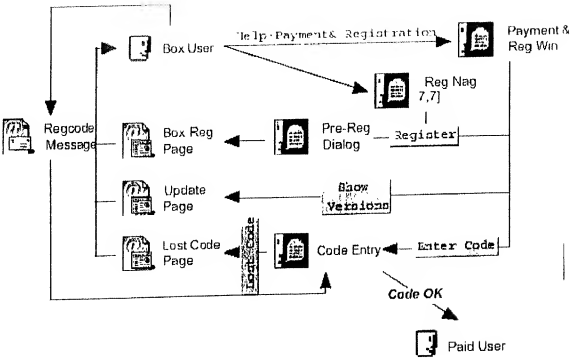

Fig. 8

D0T02T'E69982693

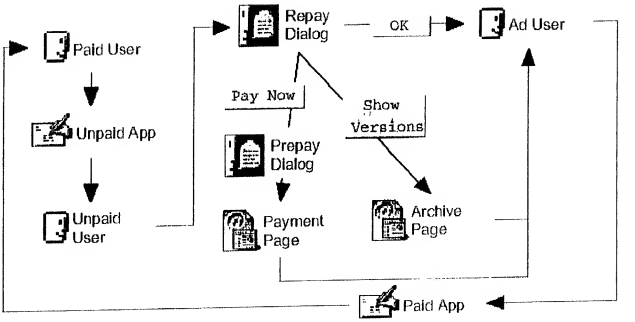

Fig. 9

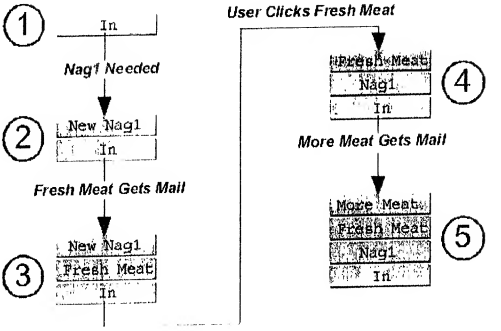
1
In
Nag1 Needed
2
New Nag1
In
Fresh Meat Gets Mail
3
New Nag1
Fresh Meat
In
User Clicks Fresh Meat
Fresh Meat
Nag1
In
4
More Meat Gets Mail
More Meat
Fresh Meat
Nag1
In
5

Fig. 10

09728693 .120100

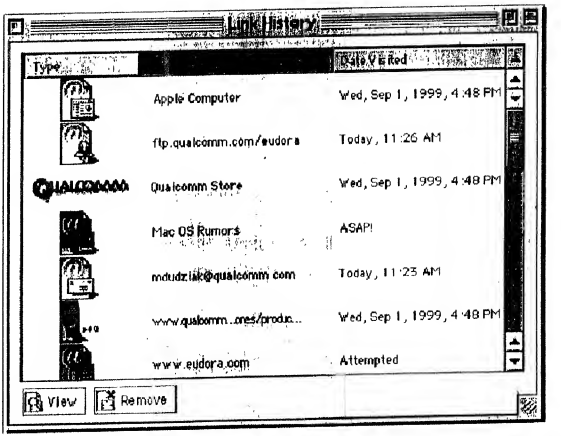

Fig. 12A

09728693 .120100

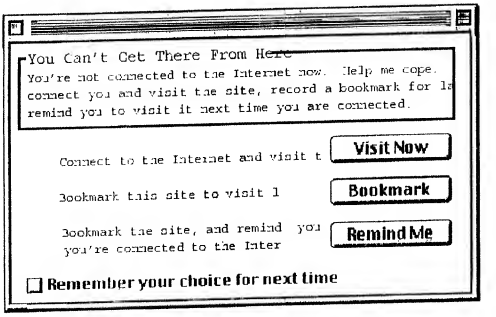

Fig. 12B

Running Items	
Average Connect Speed, Mbps	28.8
Average Ad Size, Kbytes	9.3
Number of Users	8,000,000
Number of Hours Running Subsets	2
Number Milliseconds Per User Per Hour	2
Playline Buffer Size, Bytes	500

Fig. 13A

Implications								
# of New Ads Per Users Per Day	# Seconds Downloaded per Ad	# Seconds Allocated Per Ad	8X Users Aggregate Bandwidth Check	Aggregate Bandwidth Mbps	Ad Ad Mbps / 100,000 Users	Avg Size Ad Mbps, 1000 Users	8X Users Playline Bandwidth Mbps	Playline Buffer 100,000 Users
10	26	7	87	1.4	2.4	4	1	0
15	39	10	101	1.3	3.6	5	0	1
20	52	13	135	1.7	4.8	7	0	1
25	65	16	169	2.1	6.0	9	0	1
30	78	19	202	2.5	7.2	11	0	1
35	90	23	235	2.9	8.4	12	0	2

Fig. 13B

09728693.120100

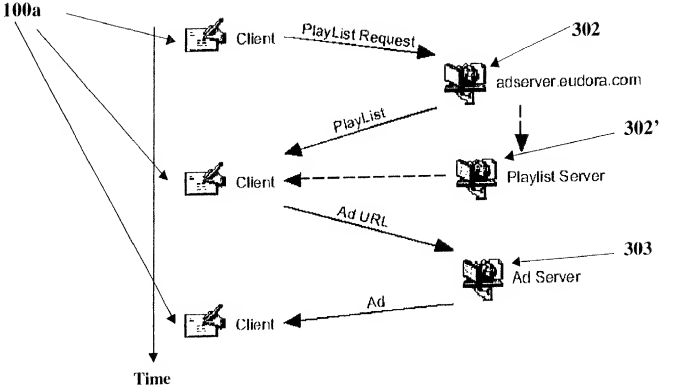

Fig. 14

```
////////////////////////////////////
// Main ad scheduler
ScheduleMain
{
// Has a new day dawned?
Do CheckForNewDay
// Are we are within the current ad's showFor?
if ( ad.thisShowTime < ad.showFor )
{
// there is nothing to be done
return
}
// At this point, we know that we need a new ad
// Perform housekeeping tasks on the old one
Do AdEndBookkeeping
// Pop out of a block if all ads on par
if ( block isn't all playlists )
{
find ad with minimum ad.numberShown
if ( ad.numberShown >= blockGoal )
set block to all playlists
}
// If we are over our quota of regular ads for the day,
// look for a runout
if ( adFaceTimeToday > faceTimeQuota )
{
Do ShowARunout
}
else
{
Do ShowARegularAd
}
}
// end ad schedule main
```

Fig. 15A

```
////////////////////////////////////
// We must perform certain tasks when the calendar day
changes.
CheckForNewDay
{if ( the calendar day has changed )
{
// Perform housekeeping tasks on the ad currently showing
Do StopShowingCurrentAd
// Runout ads are charged for a full showFor if they've been
shown
// at all on a given day. Charge any runout ads if they've
been
// shown at all.
for runout ads
{
if ( ad.thisShowTime > 0 )
{
ad.totalTimeShown += ad.showFor
ad.thisShowTime = 0
}
}
// Now, reset the counters for all ads to reflect the fact
that
// a new day has dawned.
for all ads
{
ad.numberShownToday = 0
}
// Record yesterday's facetime
// Might not literally be yesterday, be sure to use
// whatever day the app was last run on
set old current day's facetime to totalFaceTimeToday
// and reset our global regular ad facetime counter
adFaceTimeToday = 0
totalFaceTimeToday = 0
// if we were in a block, back out
set block to all playlists
}
}
// end CheckForNewDay
```

Fig. 15B

```
////////////////////////////////////
// This function shows a runout ad, and if it
// can't find one, goes to a rerun
ShowARunout
{
for runout ads
{
// has the ad been flushed?
if ( ad.flushed )
try next ad
// are we done showing this runout today?
if ( ad.numberShownToday > ad.dayMax )
try next ad // this one's used up for the day
// are we done showing this runout for ever and ever?
if ( ad.shownFor > ad.showForMax )
try next runout ad // this one's used up forever
// are we between the ad's start and end dates?
if ( ad.startDate < the current date < ad.endDate )
try next runout ad
// the ad is not supposed to run today
// do we actually HAVE the ad?
if ( ad has not been downloaded )
{
ask for ad to be downloaded
try next ad
}
// ok, we believe we should show this runout
// we are now in runout state
Do ShowAnAd
return
}
// if we haven't found a runout ad, we will go to "rerun"
state
Do ShowARerun
}
// end ShowARunout
```

Fig. 15C

```
////////////////////////////////////
// Rerun state. Look for a regular ad to rerun
ShowARerun
{
for regular ads [ in current block ]
{
// has the ad been flushed?
if ( ad.flushed )
try next ad
// is this ad recent enough to rerun?
if ( ad.lastShownDate is older than returnInterval )
try next ad
// this one is too old to rerun
// if in block, show ads only if it's their "turn"
if ( ad.numberShownToday >= blockGoal )
try next ad // need to find a friend in this block
// are we between the ad's start and end dates?
if ( ad.startDate < the current date < ad.endDate )
try next ad
// the ad is not supposed to run today
// do we actually HAVE the ad?
if ( ad has not been downloaded )
{
ask for ad to be downloaded
try next ad
}
// ok, at this point we can show this ad, but because
// we're in rerun, we don't keep the books
Do ShowAnAd
return
}
// if we get here, we have no ads to show. Punt.
return
}
// end ShowARerun
```

Fig. 15D

```
////////////////////////////////////
// Show a regular ad
ShowARegularAd
{
for regular ads [ in current block ]
{
// has the ad been flushed?
if ( ad.flushed )
try next ad
// are we done showing this ad today?
if ( ad.numberShownToday > ad.dayMax )
try next ad // this one's used up for the day
// if in block, show ads only if it's their "turn"
if ( ad.numberShownToday >= blockGoal )
try next ad // need to find a friend in this block
// are we done showing this ad for ever and ever?
if ( ad.shownFor > ad.showForMax )
try next ad // this one's used up forever
// are we between the ad's start and end dates?
if ( ad.startDate < the current date < ad.endDate )
try next ad
// the ad is not supposed to run today
// do we actually HAVE the ad?
if ( ad has not been downloaded )
{
ask for ad to be downloaded
try next ad
}
// ok, we believe we should show this ad
// we are now in regular state
Do ShowAnAd
return
}
// If we get here, we have failed to find a regular
// ad. Go to runout
Do ShowARunout
}
// end ShowARegularAd
```

Fig. 15E

```
////////////////////////////////////
// Perform necessary housekeeping when we're taking
// down an ad
AdEndBookkeeping
{
// In rerun state, we don't do any bookkeeping
if ( in RerunState )
return
// Account for at most ad.showFor seconds, provided
// we've shown the ad for at least ad.showFor seconds
// Note that this means we don't charge for time beyond
// ad.showFor seconds, which is important
if ( ad.thisShowTime >= ad.showFor )
{
ad.numberShownToday += ad.showFor
ad.shownFor++
// we do NOT reset thisShowTime here, we do it in
// AdStartBookkeeping. It actually doesn't matter where
// we do it, provided we are careful NOT to do it for
// runout ads.
}
}
// end AdEndBookkeeping
```

Fig. 15F

```
////////////////////////////////////
// Show an ad, including bookkeeping and block handling
ShowAnAd
{
// If the ad is in a block, notice that
if ( it's in a "block" playlist )
{
if ( not currently in a block )
{
find ad in block with minimum numberShown
make that our ad
set blockGoal to minimum numberShown+1
}
set current block to this playlist
}
// now do bookkeeping
Do AdStartBookkeeping
// and actually show it
Do DisplayThatAd
}
```

3217695.1 122103

Fig. 15G

```
////////////////////////////////////
// Perform housekeeping when we put up an ad
AdStartBookkeeping
{
// In rerun state, we don't do any bookkeeping
if ( in RerunState )
return
// For regular ads
if ( it's a regular ad )
{
ad.thisShowTime = 0
ad.lastShownDate = now
}
}
// end AdStartBookkeeping
```

Fig. 15H

Persistent Ads		
PlayList Request	faceTime	Used to determine how much advertising to send to client
	faceTimeLeft	Not used
PlayList Response ClientInfo	reqInterval	Relatively large: one or more days
	flush	Used. Single playlist completely specifies list of ads client should have
PlayList Response Scheduling Parameters	showForMax	Not used

Fig. 16A

Short-Lived Ads		
PlayList Request	faceTime	Not used
	faceTimeLeft	Used to determine how many ads client should receive
PlayList Response ClientInfo	reqInterval	Not used. Instead, client requests new playlist whenever ads "run low"
	flush	Not used
PlayList Response Scheduling Parameters	showForMax	Used to determine how long an ad runs

Fig. 16B

DOTOST:E69982760

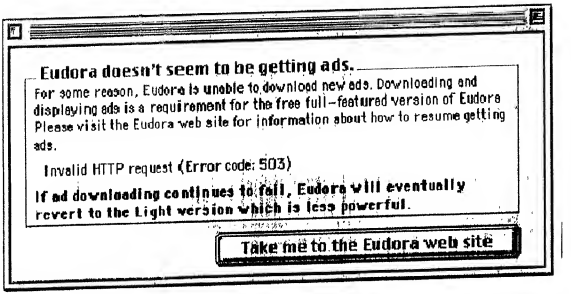

Fig. 17A

09728693 "180100

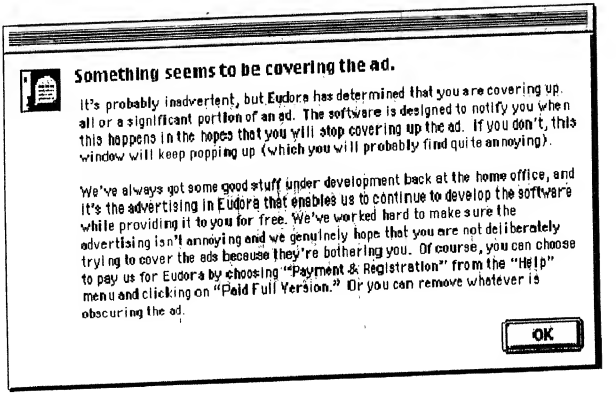

Fig. 17B

09/28/2003 12:01:00

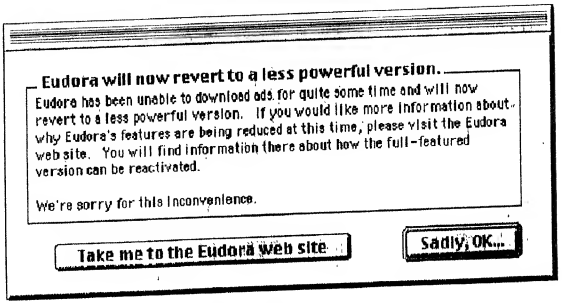

Fig. 17C

00T02T' E6982260

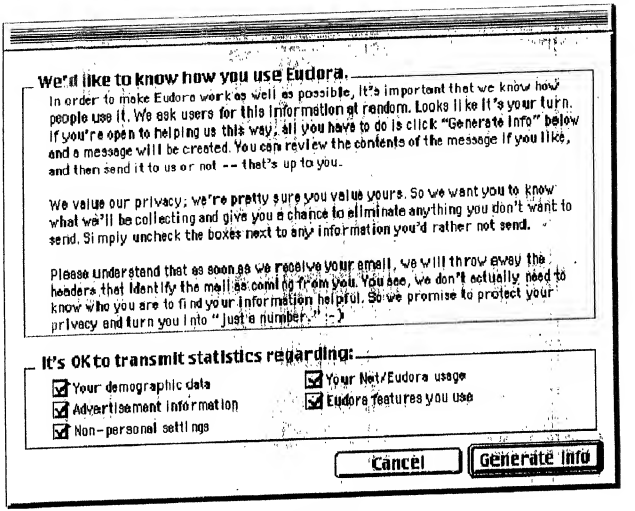

Fig. 18A

09728693.120100

Page	Applicable Query Parts																
	action	platform	product	version	distributorID	mode	realname	email	regfirst	reglast	regcode	oldReg	regLevel	profile	url	addid	topic
Payment	pay	X	X	X	X	X	X	X	X	X	X	X					
Freeware Registratio	register-free	X	X	X	X	X	X	X	X	X	X						
Adware Registration	register-ad	X	X	X	X	X	X	X	X	X	X						
Box Registrations	register-box	X	X	X	X	X	X	X	X	X	X						
Lost Code	lostcode	X	X	X	X	X	X	X	X	X	X	X					
Update	update	X	X	X	X	X							X				
Pro Update	proupdate	X	X	X	X	X							X				
Archived	archived	X	X	X	X	X											
Profile	profile	X	X	X	X	X	X	X						X			
Introduction	intro																
Support	n/a	X	X	X	X	X	X	X	X	X	X	X					
QuickTime Missing	support	X	X	X	X												no-qt
Ad Failure	support	X	X	X	X												ad-fail
Tutorial	support	X	X	X	X												tutor
FAQ	support	X	X	X	X												faq
Light Users	support	X	X	X	X												light
Search Support	support	X	X	X	X												search
Newsgroups	support	X	X	X	X												usenet

Fig. 19

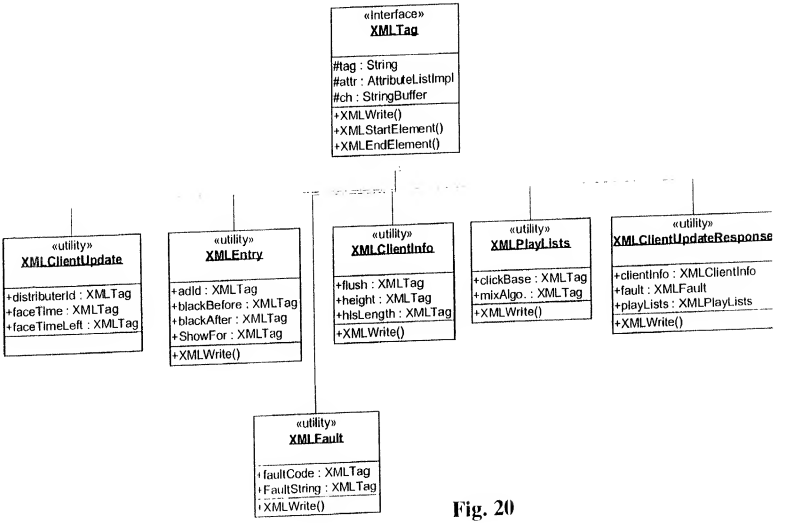
«Interface»
XMLTag
#tag : String
#attr : AttributeListImpl
#ch : StringBuffer
+XMLWrite()
+XMLStartElement()
+XMLEndElement()
«utility»
XMLClientUpdate
+distributerId : XMLTag
+faceTime : XMLTag
+faceTimeLeft : XMLTag
«utility»
XMLEntry
+adId : XMLTag
+blackBefore : XMLTag
+blackAfter : XMLTag
+ShowFor : XMLTag
+XMLWrite()
«utility»
XMLClientInfo
+flush : XMLTag
+height : XMLTag
+hlsLength : XMLTag
+XMLWrite()
«utility»
XMLPlayLists
+clickBase : XMLTag
+mixAlgo. : XMLTag
+XMLWrite()
«utility»
XMLClientUpdateResponse
+clientInfo : XMLClientInfo
+fault : XMLFault
+playLists : XMLPlayLists
+XMLWrite()
«utility»
XMLFault
+faultCode : XMLTag
+FaultString : XMLTag
+XMLWrite()

Fig. 20

00702E7 E6982760

% The list of available ads advantageously can be built from the following query:

ads = dbCon.prepareStatement("SELECT * FROM ads WHERE StartDate <= today AND endDate >= today + 30 AND AdType = "I" AND AdStatus = "A" AND ImpressionsServed < Impressions ORDERD BY ImpressionsServed ASC);

run out ads = dbCon.prepareStatement("SELECT * FROM ads WHERE StartDate <= today AND endDate >= today + 30 AND AdType = "R" AND AdStatus = "A" AND ImpressionsServed < Impressions ORDERD BY ImpressionsServed ASC);

% The time required to deliver the ads advantageously can be calculated in the following manner.

face time left for today [seconds] = faceTime[today] – faceTimeUsedToday

(Comment: Face time left for today is the number of secondes the servlet can use to deliver special ads today.)

predict face time [seconds] = SUM(faceTime[tomorrow] , faceTime[tomorrow + 1] , ... faceTime[tomorrow + reqInterval])

(Comment: Predict face time is the number of seconds the servlet predicts the user is going to have.)

goal show time left [seconds] = predict face time – faceTimeLeft

(Comment: Goal show time left is the number of seconds that the software provider needs to fill with ads.)

Fig. 21A

```
% Targeting
    while (face time left for today ) {
        if ad is not in the history {
            select ad [according to target = today]
            face time left for today -= ad.showFor
        }
        next ad
}

while (Goal show time left ) {
    if ad is not in the history {
        select ad [according to target]
        goal show time left -= ad.showFor
    }
    next ad
}


Default values:
    reqInterval = 1 day.
    facetime = 30 minutes
    faceTimeQuota is ?
    histLength = 31 days
```

Fig. 21B

09782693.120100

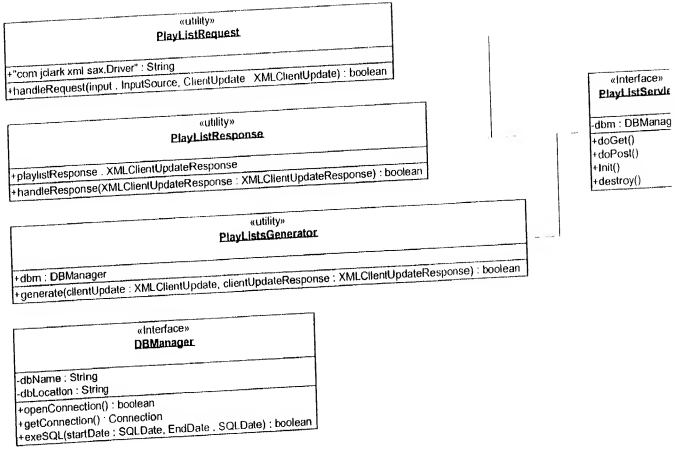

Fig. 22

00T0ET'S69888760

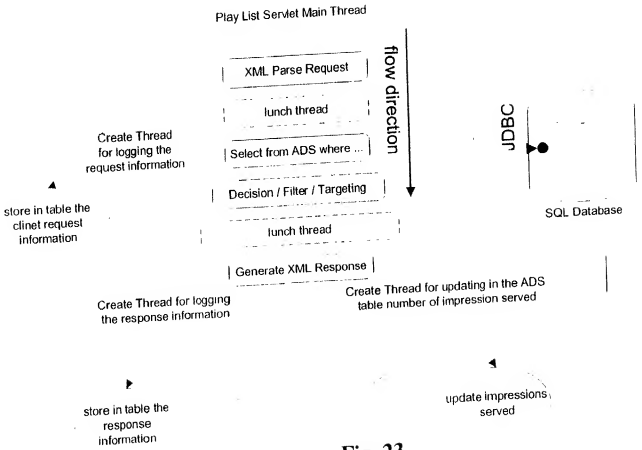

Fig. 23